Visual Analytics Applied to Image Analysis

Rauber, Paulo

6

VISUALIZING TIME-DEPENDENT DATA USING PROJECTIONS

In the previous chapter, we attempted to visualize *evolving* (in a broad sense, time-dependent) high-dimensional data using a standard dimensionality reduction technique. In this chapter, we will discuss the drawbacks of this approach in more detail, and propose an alternative.

Time-oriented data visualization is a widely researched subject. According to Aigner *et al.* [1], current techniques can be categorized as abstract or spatial, univariate or multivariate, linear or cyclic, instantaneous or interval-based, static or dynamic, and two or three-dimensional. Our work is concerned with abstract, multivariate, and instantaneous time-oriented visualization.

We define a time-dependent dataset as a sequence of datasets captured at particular time steps. In such a sequence, each dataset is a sequence of observations, and each observation has a corresponding observation across time steps. In simple terms, each observation *evolves* with time (or any other discrete parameter).

Consider the task of visualizing a time-dependent dataset. If a dimensionality reduction (DR) technique is applied independently for each time step, the resulting sequence of projections may present variability that does not reflect significant changes in the *structure* of the data. We refer to this issue as *temporal incoherence*, which significantly impairs the visualization of temporal trends. In this chapter, we will show that this issue affects t-SNE [99], a technique whose importance was already established in the previous chapters. Furthermore, temporal incoherence will affect any DR technique that is sensitive to relatively small changes in their inputs [49].

In this context, we also propose dynamic t-SNE: an adaptation of t-SNE that allows a controllable trade-off between temporal coherence and spatial coherence (defined as preservation of structure at a particular time step). Previous work on this trade-off has been restricted to the context of dynamic graph drawing [92, 158], even though there are many examples of time-dependent high-dimensional data visualizations based on DR [3, 13, 74]. As will become clear, our approach can be easily extended to other optimization-based DR techniques.

This chapter is organized as follows. Section 6.1 briefly reviews our notation and t-SNE. Section 6.2 explains the necessity for a controllable bias towards temporal coherence, and presents our proposed solution.

This chapter is based on the following publication:
P. E. Rauber, A. X. Falcão, and A. C. Telea. Visualizing time-dependent data using dynamic t-SNE. In *EuroVis Short Papers*, 2016.

Section 6.3 presents a preliminary evaluation of this proposal. Finally, Section 6.4 summarizes our contributions and suggests future work.

6.1 T-SNE

A dataset $\mathcal{D} = \mathbf{x}_1, \ldots, \mathbf{x}_N$ is a sequence of observations, which are D-dimensional real vectors. The goal of t-SNE is to compute a sequence of points (projection) $\mathcal{P} = \mathbf{p}_1, \ldots, \mathbf{p}_N$ where the *neighborhoods* from $\mathcal{D}$ are preserved, considering that each $\mathbf{p}_i \in \mathbb{R}^d$ corresponds to $\mathbf{x}_i \in \mathbb{R}^D$. Typically, $d = 2$ and $D \gg d$.

T-SNE aims at minimizing a particular cost C with respect to $\mathcal{P}$. For our purposes, it suffices to note that C heavily penalizes placing *neighbors* in $\mathcal{D}$ far apart in $\mathcal{P}$. We refer to Sec. 2.6.4 for more details.

The cost C is usually minimized with respect to $\mathcal{P}$ by (momentum-based) gradient descent: from an arbitrary initial $\mathcal{P}$, for a number of iterations, each $\mathbf{p}_i \in \mathcal{P}$ is moved in the direction $-\nabla_{\mathbf{p}_i} C$.

As we explained in Sec. 2.6.4, the gradient $\nabla_{\mathbf{p}_i} C$ of C with respect to a point $\mathbf{p}_i \in \mathcal{P}$ can be interpreted as a linear combination of vectors pointing in the direction $\mathbf{p}_i - \mathbf{p}_j$, for every j. Each vector $\mathbf{p}_i - \mathbf{p}_j$ is also weighted by whether $\mathbf{p}_j$ should be moved closer to $\mathbf{p}_i$ to preserve neighborhoods from $\mathcal{D}$, and by whether $\mathbf{p}_j$ is currently close to $\mathbf{p}_i$.

6.2 DYNAMIC T-SNE

Consider the task of creating a sequence of projections $\mathcal{P}[1], \ldots, \mathcal{P}[T]$ for a (sequence of datasets) time-dependent dataset $\mathcal{D}[1], \ldots, \mathcal{D}[T]$, where each $\mathbf{x}_i[t] \in \mathcal{D}[t]$ corresponds to $\mathbf{x}_i[t+1] \in \mathcal{D}[t+1]$. Although we will say that the sequence of datasets represents a *time*-dependent process, this task is meaningful whenever there is correspondence between observations at different *steps*.

We will let $C[t]$ denote the usual t-SNE cost for dataset $\mathcal{D}[t]$ and projection $\mathcal{P}[t]$, as defined in Sec. 2.6.4. It is possible to apply t-SNE individually for each dataset in a sequence using at least four different strategies:

1. Initializing $\mathcal{P}[t]$ independently and randomly, for all t.
2. Initializing $\mathcal{P}[t]$ with the same random sequence, for all t.
3. Initializing $\mathcal{P}[1]$ randomly, and $\mathcal{P}[t+1]$ with the $\mathcal{P}[t]$ that results from minimizing $C[t]$, for all $t > 1$, or reversely.
4. Combining datasets from all time steps into a single dataset $\mathcal{D}$, and computing a single projection $\mathcal{P}$.

However, each of these strategies has significant drawbacks.

Strategies 1 and 2 often result in a sequence of projections with major changes in positioning of corresponding points in adjacent time

steps (temporal incoherence). This issue cannot be corrected by rigid transformations (*e.g.*, rotations, translations), is misleading, and makes tracking the evolution of the data more challenging (see Sec. 6.3.1).

Strategy 3 is viable in some cases. However, it lacks a mechanism to enforce temporal coherence after initialization. At the other extreme, the initial bias may be difficult for gradient descent to overcome, because of the diminished effect on $\nabla_{\mathbf{p}_i[t]} C[t]$ of a point that is distant from $\mathbf{p}_i[t]$. These two issues also affect the (unlisted) strategy employed in the previous chapter, where we initialized $\mathcal{P}[t]$ with a projection created for $\mathcal{D}[1]$, for all t (including $t = 1$).

Furthermore, returning to strategy 3, because t-SNE usually takes many iterations to converge, the optimization of $C[t]$ starts at a likely *advantaged* state when compared to the optimization of $C[t']$, for all $t' \ll t$. In this case, the evolution due to the optimization process can be mistaken for temporal evolution.

As an extreme example, consider a particular sequence of 100 *identical* datasets, each with 2000 observations in $\mathbb{R}^{512}$. Figure 6.1 shows some projections that result from strategy 3, which are clearly misleading. Notice how there is significant *apparent* evolution between time steps 1 and 50 (10^3 and 5×10^4 gradient descent iterations, respectively). In fact, the configuration still changes between 5×10^4 and 10^5 iterations, albeit more slowly. Running t-SNE for this many iterations (for each projection) is impractical, and tweaking the parameters to achieve faster convergence is not trivial. Although it suffices to realize that there is no actual temporal evolution in this time-dependent dataset, the experimental details are described in Sections 6.3 and 6.3.2.

In summary, the major issue with strategy 3 is the lack of control over how the optimization is biased.

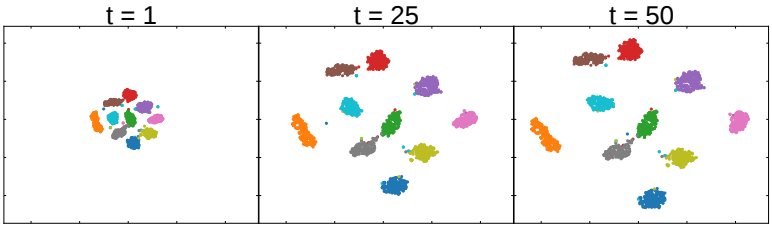

Figure 6.1: Strategy 3 results on a sequence of *identical* datasets (last CNN hidden layer fixed at epoch 1, MNIST test subset).

Strategy 4 can be dismissed in many cases. Firstly, when the distance matrix for $\mathcal{D}$ and all σ_i are given as inputs, and the target dimension d is seen as a constant, t-SNE has time complexity $O(N^2)$ for N observations. Thus, strategy 4 quickly becomes intractable. Secondly, it also introduces significant *clutter*, which cannot be eliminated by filtering points per time step, since that introduces misleading void spaces. Fi-

nally, depending on context, combining *structures* across different steps may be inappropriate.

Dynamic t-SNE, our proposal, is an alternative that overcomes the drawbacks of the previous strategies. The dynamic t-SNE cost C tries to preserve the neighborhoods from $\mathcal{D}[t]$ in $\mathcal{P}[t]$, for each t, but also penalizes each point for unnecessarily moving between time steps. This new cost introduces a hyperparameter $\lambda \geq 0$ that *controls* the bias for temporal coherence, and is given by

$$C = \sum_{t=1}^{T} C[t] + \frac{\lambda}{2N} \sum_{i=1}^{N} \sum_{t=1}^{T-1} || \mathbf{p}_i[t] - \mathbf{p}_i[t+1] ||^2. \tag{6.1}$$

Intuitively, each point is penalized in proportion to the total squared length of the line segments formed by its movement through time. This penalty is similar to the one proposed by Leydesdorff and Schank [92] for dynamic graph drawing using multidimensional scaling (MDS).

Although it would be possible to penalize each movement in $\mathbb{R}^d$ in proportion to the corresponding movement in $\mathbb{R}^D$, that would have undesirable consequences. Firstly, supposing large variance in high-dimensional movement, it could make the choice of λ considerably more difficult. Secondly, any transformation that moved observations significantly while preserving most pairwise distances would justify significant changes in the projection. This is undesirable because the t-SNE cost depends solely on distances, which makes a projection convey only relative positioning. Despite these issues, we believe that such alternatives should be investigated in future work.

It is easy to show that the gradient $\nabla_{\mathbf{p}_i[t]} C$ of C with respect to a point $\mathbf{p}_i[t] \in \mathcal{P}[t]$ is given by

$$\nabla_{\mathbf{p}_i[t]} C = \nabla_{\mathbf{p}_i[t]} C[t] + \frac{\lambda}{N} \mathbf{v}_i[t], \tag{6.2}$$

where $\nabla_{\mathbf{p}_i[t]} C[t]$ is the usual t-SNE cost gradient (with respect to $\mathbf{p}_i[t]$) when the dataset $\mathcal{D}[t]$ is considered separately, and $\mathbf{v}_i[t]$ is given by

$$\mathbf{v}_i[t] = \begin{cases} 2\mathbf{p}_i[t] - (\mathbf{p}_i[t-1] + \mathbf{p}_i[t+1]) & \text{if } 1 < t < T, \\ \mathbf{p}_i[t] - \mathbf{p}_i[t+1] & \text{if } t = 1, \\ \mathbf{p}_i[t] - \mathbf{p}_i[t-1] & \text{if } t = T. \end{cases} \tag{6.3}$$

Just as $\nabla_{\mathbf{p}_i[t]} C[t]$, each vector $\mathbf{v}_i[t]$ also has a geometrical interpretation. For $1 < t < T$, the vector $\mathbf{v}_i[t]$ has *opposite* direction to any vector that points from $\mathbf{p}_i[t]$ to the midpoint between $\mathbf{p}_i[t-1]$ and $\mathbf{p}_i[t+1]$. Thus, in gradient *descent*, the parameter λ controls the trade-off between moving each $\mathbf{p}_i[t]$ in a direction that tries to preserve neighborhoods from $\mathcal{D}$, and moving each $\mathbf{p}_i[t]$ in a direction that minimizes the total squared length of line segments in the polyline $(\mathbf{p}_i[t-1], \mathbf{p}_i[t], \mathbf{p}_i[t+1])$.

6.3 EVALUATION

We implemented t-SNE and dynamic t-SNE in Python, using Theano [9], Numpy [152], and scikit-learn [121] . Theano allows writing mathematical expressions that can be automatically translated into optimized (CPU or GPU) code and evaluated. Our implementation uses automatic differentiation, which can be highly valuable for adapting t-SNE to a particular application. For instance, altering the symbolic expression that defines the cost does not require manually finding (possibly involved) partial derivatives analytically, nor changing the optimization process. Dynamic t-SNE requires roughly the same computational time as executing t-SNE independently for each time step (Strategies 1-3). Using an *Intel i7-2600* at 3.4 GHz with a GeForce GTX 590, both (GPU) implementations require approx. 6 minutes *per time step* for the time-dependent dataset in Sec. 6.3.1.

The remainder of this section presents our preliminary experimental evaluation of dynamic t-SNE. The implementation details and hyperparameter choices are very similar to those of publicly available implementations [151]. We use momentum-based gradient descent for minimizing C, with a learning rate $\eta = 2400$ and momentum coefficient $\mu = 0.5$, which change to $\eta = 250$ and $\mu = 0.8$ at iteration 250. The optimization is run for 1000 iterations, with a perplexity $\kappa = 70$. We sample the initial coordinates of each point from a Gaussian distribution with zero mean and standard deviation 10^{-4}. The binary search for each σ_i lasts 50 iterations. For dynamic t-SNE, every projection $\mathcal{P}[t]$ is initialized equally.

6.3.1 *Multivariate Gaussians*

We created the multivariate Gaussians dataset specifically as a controlled experiment for dynamic t-SNE. Firstly, we sample 200 observations from each of 10 distinct (isotropic) 100-dimensional multivariate Gaussian distributions with variance 0.1. We combine the resulting 2000 observations into a single dataset $\mathcal{D}[1]$. Each multivariate Gaussian has a distinct mean, which is chosen uniformly between the standard basis vectors for $\mathbb{R}^{100}$. Given $\mathcal{D}[t]$, the dataset $\mathcal{D}[t+1]$ is created as follows. Each observation $\mathbf{x}[t+1] \in \mathcal{D}[t+1]$ corresponds to an observation $\mathbf{x}[t] \in \mathcal{D}[t]$ moved 10% of the remaining distance closer to the mean of its corresponding multivariate Gaussian. In simple terms, each of the 10 clusters becomes more *compact* as t increases. We consider $T = 10$ datasets.

The sequence of images in Fig. 6.2a shows dynamic t-SNE results for $\lambda = 0$, which corresponds to strategy 2 (as defined in Sec. 6.2). Each point $\mathbf{p}_i[t]$ is colored, for illustration purposes, according to the distri-

Available in `https://github.com/paulorauber/thesne`.

bution from which $\mathbf{x}_i[1]$ was sampled. Notice the large variability in *visual cluster* positioning between time steps, even after the clusters become well-defined. Because the process that originates the data simply makes the clusters gradually more compact, this variability is misleading. We preserve the scatterplot scale between time steps, which is also a significant source of variability.

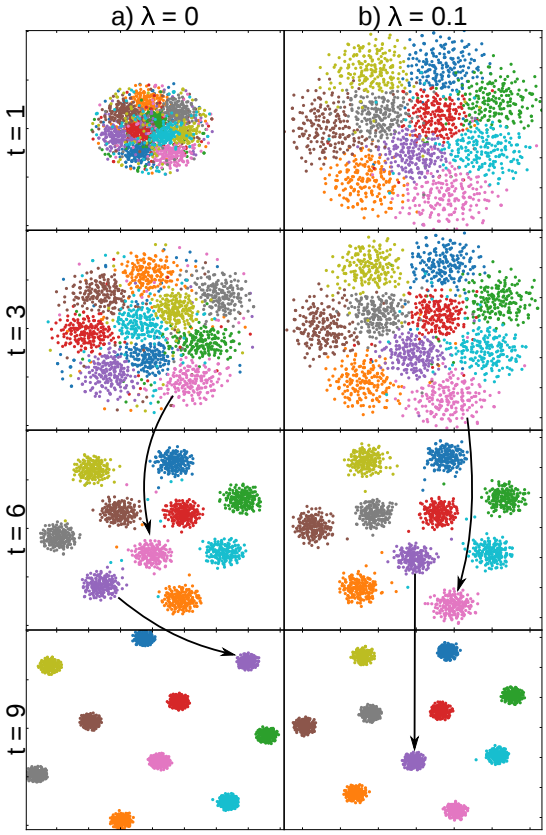

Figure 6.2: Dynamic t-SNE results on Multivariate Gaussians.

In comparison, consider the results shown in Fig. 6.2b, for $\lambda = 0.1$. Notice how each cluster stays at a similar relative position during all steps, and only becomes more compact in later steps. When the projections are inspected step by step, it becomes easier to notice the movement of projection *outliers*, which is obscured when $\lambda = 0$.

Because each point is penalized for moving between projections, clear *visual separation* between clusters in later projections is also able to induce better separation in earlier projections. In simple terms, given a similar spatial coherence in two alternative projections for time step t, the projection that is more temporally coherent with the projection for

time $t + 1$ is preferred by the cost function. There is a trade-off: a large λ will induce unwanted bias, whereas a small λ will cause misleading temporal incoherence. The major benefit of dynamic t-SNE is precisely the control over this trade-off. Although the choice of λ depends on context, we recommend first comparing $\lambda = 0$ with the results of an arbitrary low value.

6.3.2 *Hidden layer activations*

The time-dependent dataset $\mathcal{D}[0], \ldots, \mathcal{D}[T]$ considered in this section is composed of neural network activation sets, which we introduced in the previous chapter. Recall that an activation vector $\mathbf{x}[t] \in \mathcal{D}[t]$ is a D-dimensional real vector that represents the outputs of D neurons in a particular layer of an artificial neural network given a particular input. Such activation vector can be seen as an alternative representation of the input, learned by the network through an optimization process. As we have shown in the previous chapter, visualizing activation vectors allows valuable insight into how a network learns and operates, which is considered highly valuable by practitioners.

In this particular case, each network input belongs to a subset of 2000 test images from the SVHN dataset [113], a traditional image classification benchmark, and is assigned to one of ten classes (according to the digit seen in the image), which we use to color the projections (see Secs. 5.3 and 5.4.2 for more details).

For each t, an activation $\mathbf{x}[t] \in \mathcal{D}[t]$ is a 512-dimensional real vector, and corresponds to the last hidden layer activation of a convolutional neural network (CNN) after t epochs of training (given a particular input image). The time-dependent dataset represents the evolution of the learned representations through 100 epochs. Earlier in the text, the projections shown in Fig. 6.1 correspond to a similar dataset based on 2000 MNIST [87] test images, and 100 copies of the same dataset after one training epoch.

Figures 6.3a and 6.3b compare the results of dynamic t-SNE for $\lambda = 0$ and $\lambda = 0.1$, respectively. Notice that the projections for step $t = 0$, which correspond to network activations before training, are noticeably different from those that follow. Clearly, the early epochs of training have a significant effect on the learned representations, which coincides with most of the increase in validation accuracy (not shown). Although both sequences show significant variation between steps $t = 25$ and $t = 100$, the remarkable distinction is that the projections change *smoothly* when $\lambda = 0.1$. For an example, compare the transition between steps 24 and 25 in Figs. 6.3a and 6.3b. This phenomenon can be seen consistently through the whole sequence. The visual separation between clusters does not seem to improve considerably after the early epochs, although it is hard to state whether there is significant variability in the structure of the data. Because $\lambda = 0.1$ does not seem to intro-

duce a misleading bias in comparison to $\lambda = 0$, more evidence could be obtained by increasing λ even further.

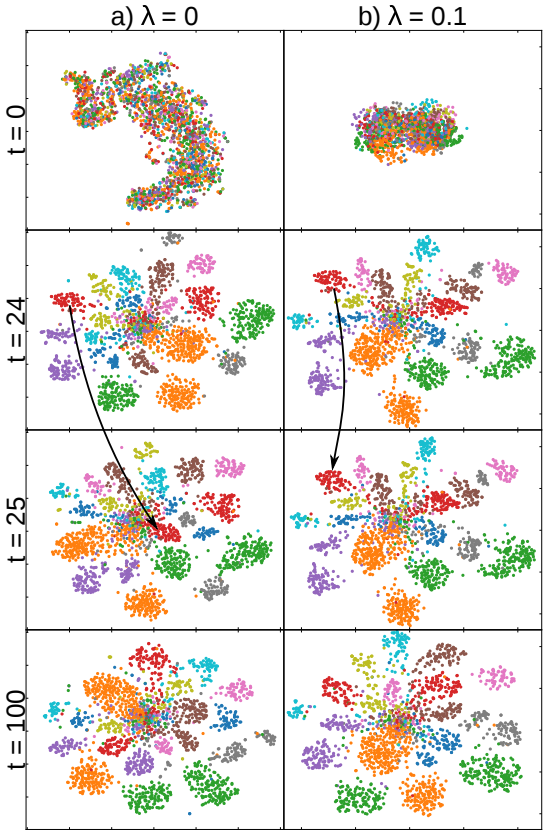

Figure 6.3: Dynamic t-SNE results on SVHN CNN.

6.4 CONCLUSION

In this chapter, we have shown how dynamic t-SNE can be applied to create sequences of projections with increased temporal coherence, which facilitates tracking the evolution of high-dimensional time-dependent data. The main advantage of dynamic t-SNE over t-SNE is the control over the trade-off between temporal coherence (between successive projections) and spatial coherence (with respect to high-dimensional neighborhoods). This control depends on a single hyperparameter λ, which has a simple interpretation, and does not introduce a significant computational overhead. This approach can be easily adapted for other optimization-based DR techniques. Our prelim-

inary experiments show promising results in eliminating unnecessary variability between projections.

Although we implemented dynamic t-SNE as an adaptation of *traditional* t-SNE, the Barnes-Hut approximation is significantly more computationally efficient [150]. Future works that employ dynamic t-SNE for large datasets should consider a similar optimization. The current implementation has the advantage of being highly flexible with respect to cost functions.